Homework 5

Due by **11:59pm, Thursday, October 11, 2018**.

Make sure you follow all the homework policies (,,/..//policies/hw-policy.html).

All submissions should be done via Autolab (../autolab.html).

Sample Problem

The Problem

Extend the topological ordering algorithm we saw in class so that, given an input directed graph G, it outputs one of two things: (a) a topological ordering, thus establishing that G is a DAG, or (b) a cycle in G, thus establishing that G is not a DAG.

The running time of your algorithm should be $O(m+n)$ for a directed graph with n nodes and m edges.

Click here for the Solution

Submission

You will **NOT** submit this question. This is for you to get into thinking more about designing algorithms on graphs.

Question 1 (Programming Assignment) [30 points]

</> Note

This assignment can be solved in either Java, Python or C++ (you should pick the language you are most comfortable with). Please make sure to look at the supporting documentation and files for the language of your choosing.

The Problem

We are given an undirected graph G with n nodes. The graph is formatted as an adjacency list, meaning that for each node, u, we can access all of u's neighbors. The goal is to output a single cycle from G. If there is no cycle, then return an empty list.

> ### Note
> Note that as we mentioned in class, a cycle in an undirected graph (as is the case in this problem) needs to have **at least** three nodes in it.

Input

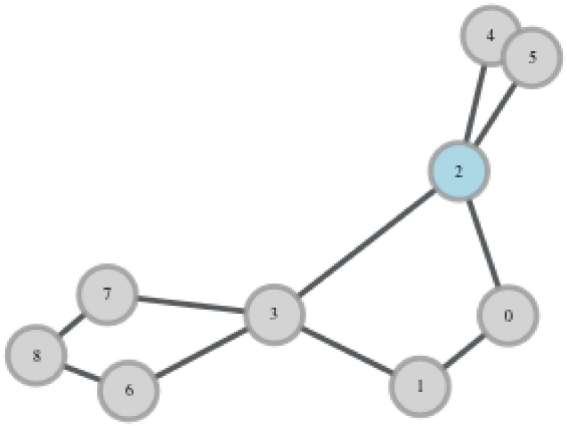
The input file is given as an adjacency list for graph G (we assume that the set of vertices is $\{0, 1, \ldots, n-1\}$). The adjacency list assumes that the current node is the index of the line under consideration. For instance line 0 of the input file has the list of all nodes adjacent to node 0.

```
u1 u4 u6          <- All nodes that share edges with u0
u3 uu             <- All nodes that share edges with u1
u0                <- All nodes that share edges with u2
.
.
.
u0 u4 u2 u7       <- All nodes that share edges with un-1
```

For example:

```
1 2               <- Node 0 shares edges with nodes 1 & 2
0 3               <- Node 1 shares edges with nodes 0 & 3
0 3 4 5           <- Node 2 shares edges with nodes 0, 3, 4 & 5
1 2 6 7           <- Node 3 shares edges with nodes 1, 2, 6 & 7
2                 <- Node 4 shares edges with node 2
2                 <- Node 5 shares an edge with node 2
3 8               <- Node 6 shares an edge with nodes 3 & 8
3 8               <- Node 7 shares edges with nodes 3 & 8
6 7               <- Node 8 shares edges with nodes 6 & 7
```

If we draw the graph using the above input, it will look like this:



Output

The output is a list of nodes that will create a cycle. If our cycle is $[u_0, u_1, \ldots u_m]$ where m is the length of our cycle then we assume there exists an edge between every adjacent element in the list and that there is an edge between u_m and u_0.

```
[u₀ u₁ u₂... uₘ]        <- u₀ shares an edge with uₘ & u₁
                           u₁ shares an edge with u₀ & u₂
                           u₂ shares an edge with u₁ & u₃
                           .
                           .
                           .
                           uₘ shares an edge with uₘ₋₁ & u₀
```

For example, using the example input from above:

```
[0, 1, 3, 2]                <- Node 0 shares an edge with 1 & 2
                               Node 1 shares an edge with 0 & 3
                               Node 3 shares an edge with 1 & 2
                               Node 2 shares an edge with 3 & 0
```

</> Note

There is **no** guarantee that the graph has only one connected component. Also the input graph might **not** have any cycle in it.

</> Note

It is possible that the input graph might have multiple cycles in which case the given outputs might not match yours (even if your algorithm is correct).

Hint

The best possible algorithm for this problem that we are aware of runs in time $O(m+n)$ where m in the total number of edges, and n is the total number of nodes.

! Note

Both the input and output parsers in each of the three languages are already written for you. Note that you have to work with the input data structures provided (which will come pre-loaded with the data from an input file).

! Addition is the only change you should make

Irrespective of what language you use, you will have to submit just one file. That file will come pre-populated with some stuff in it. You should **not** change any of those things because if you do you might break what the grader expects and end up with a zero on the question. You should of course add stuff to it (including helper functions and data structures as you see fit).

Java | Python | C++

Download Java Skeleton Code (HW5Java.zip)

Directory Structure

```
├── src
│   ├── ub
│       ├── cse
│           ├── algo
│               ├── Driver.java
│               ├── Solution.java
├── testcases/
│   ├── input1.txt
│   ├── input2.txt
│   ├── input5.txt
├── outputs/
│   ├── output1.txt
│   ├── output2.txt
│   └── output5.txt
```

You are given two coding files: `Driver.java` and `Solution.java`. `Driver.java` takes the input file, parses it, creates an instance of the class and prints the result in standard out. You only need to update the `Solution.java` file. You may write your own helper methods and data structures in it.

The testcases folder has 3 input files and the outputs folder has their corresponding output files for your reference. We will use these three input files (and seven others) in our autograding. The output files provided are not the ONLY solutions. They only represent an example of a cycle from each of the inputs.

Method you need to write:

```
5.
/**
 * This method must be filled in by you. You may add other
 * but they must remain within the Solution class.
 * @return A list of integers corresponding the unique nod
 */
public ArrayList<Integer> findCycle() {
        return new ArrayList<>();
}
```

The `Solution` class has 1 instance variable:

- `graph` which is of type `HashMap<Integer, ArrayList<Integer>>` where the key is the node id and the value is an arraylist of adjacent nodes.

The output is an `ArrayList<Integer>` . In particular, the list of unique node ids such that these nodes create a cycle. If there is no cycle, output an empty arraylist.

Compiling and executing from command line:

Assuming you're in the same directory level as `src/` . Run `javac src/ub/cse/algo/*.java` to compile.

To execute your code on `input1.txt` , run `java -cp "src" ub.cse.algo.Driver testcases/input1.txt` . The output array will be printed to `stdout` .

Submission

You only need to submit `Solution.java` to Autolab.

Grading Guidelines

We will follow the usual grading guidelines for programming questions (../../policies/hw-policy.html#grading).

> **! For those of you who are feeling a little ambitious**
>
> For the top 3 submissions in the scoreboard in both Java and Python and for the top 2 submissions in the scoreboard in C++, we are offering 3 bonus points. But be warned! You should not be spending too much time on this. We rather you work on Questions 2 and 3 below.

Question 2 (Helping out Akash and Kiran) [45 points]

> ### The Problem
>
> Akash Rudra has always been interested in moths and butterflies. Also his sister, Kiran, is obsessed with caterpillars. On a recent field trip, Akash and Kiran collected n caterpillars and they want to identify each caterpillar as something that will grow into either a butterfly or a moth. Akash and Kiran are new to studying caterpillars so they cannot do this directly.

However, they can do the following: given a pair of caterpillars (i, j) for $i \neq j$, they can judge them as `same` (meaning they believe that both are either going to become moths or both are going to become butterflies) or `different` (meaning they believe exactly one of the two caterpillars will grow into a moth and the other into a butterfly) or `ambiguous` (meaning they are not sure and does not want to judge them as same or different). Assume that there are m pairs of caterpillars that they can label as same or different (of course $m \leq \binom{n}{2}$). Akash and Kiran are too young to take the algorithms course so they come to you for help: they want to know whether their judgments are `consistent` : i.e. is there a labeling of the n caterpillars as either butterfly or moth that is consistent with their judgments and if you can design an algorithm for them that they can run to check whether their judgments are consistent.

More precisely, the input to your algorithm is the set of n caterpillars and m definitive judgments (a definitive judgment is a pair of caterpillar $i \neq j$ such that (i, j) is either judged as `same` or `different`)-- the pairs that are not definitive judgments are `ambiguous` . Your algorithm must decide if there exists a labeling of the n caterpillars (i.e. each caterpillar is labeled as either `moth` or `butterfly`) that is consistent: i.e. for each definitive judgment (i, j) that says `same` , the labels of i and j must match and for each definitive judgment (i, j) that says `different` , the labels of i and j are different. (For a pair (i, j) that has a judgment of `ambiguous` , the labeling can assign i and j in any arbitrary manner.)

Note

You can assume that the input is a list where each entry is a pair of the form $\langle (i, j), \ell \rangle$, where ℓ is one of the labels in { `same` , `different` }. Note that if a pair (i, j) is not present in the input list then it is by default labeled as `ambiguous` .

Here are the two parts of the problem:

- Part `(a)` : Assume you are given a consistent labeling for a set of n specimens and m definitive judgements. Then argue that the labeling one gets by "flipping" each label (i.e. every specimen that was labeled `moth` before is now labeled `butterfly` and every specimen that was labeled `butterfly` before is now labeled `moth`) is also consistent.
- Part `(b)` : Akash and Kiran are still young so to make sure they do not get frustrated, design a linear $O(m + n)$ time algorithm that decides whether given the n specimen and m definite judgements whether there exists a consitent labeling. Note that if there is a consistent labeling, your algorithm does not have to output one.

Hint

It *might* be useful to represent the input as a graph and run one of the connectivity algorithms we have seen in class. Also in one way of representing the "relationships," there might be two kinds of edges: i.e., it might be useful to "label" the edges. As usual, feel free to ignore this hint and come up with an algorithm from scratch.

⊘ Common Mistake

A common mistake is to mis-interpret the meaning of the labels `same` , `different` and `ambiguous` . In particular, note that your algorithm has to decide if there is a labeling of the caterpillars that is consistent with the labelings of all the pairs.

Walkthrough video

Submission

You need to submit **one PDF** file to Autolab. We recommend that you typeset your solution but we will accept scans of handwritten solution-- you have to make sure that the scan is legible.

> **!** PDF only please
>
> Autolab might not be able to display files in formats other than PDF (e.g. Word cannot be displayed). **If Autolab cannot display your file, then you will get a zero (0) on the entire question. Note that Autolab will NOT give an error message if you submit non-PDF file, so it is YOUR responsibility to make sure you submit in the correct format.** Also the file size has to be **at most 3MB**.

Grading Guidelines

We will follow the usual grading guidelines for non-programming questions (../../policies/hw-policy.html#grading). Here is a high level grading rubric specific to part `(a)` of this problem:

1. `Proof Idea` : `10` points.

and here is the high level grading rubric for part `(b)` :

1. `Algorithm idea` : `7` points.
2. `Algorithm details` : `7` points.

3. `Proof of correctness idea` : `7` points for arguing correctness of the algorithm.
4. `Proof details` : `7` points.
5. `Runtime analysis` : `7` points, i.e. show why it runs in $O(m+n)$ time.
6. `Note` : If you only attempt part `(b)` , then your total score for part `(b)` will be used as your score for part `(a)` (scaled to out of a maximum of `10` points of course).

> **! Note**
>
> **If you do not have separated out proof idea, algorithm idea, proof details, algorithm details and runtime analysis for part `(b)` , you will get a zero(0) in the entire part irrespective of the technical correctness of your solution**.

> **Templates**
>
> Download LaTeX template. (latex-template.zip)
>
> Download Microsoft Word template. (word-template.zip)

> **! Note**
>
> Your must explicitly list your sources and collaborators when you upload your submission to Autolab. Note that there are only five allowed sources (../../policies/hw-policy.html). If you have used a source that is not allowed, please do **not** submit your homework. If you did not consult any source or did not collaborate with anyone just say `None` .

Question 3 (Listing Triangles) [25 points]

The Problem

A `triangle` in a graph $G=(V,E)$ is a 3-cycle; i.e. a set of three vertices $\{u,v,w\}$ such that $(u,v),(v,w),(u,w) \in E$. (Note that G is undirected.) In this problem you will design a series of algorithms that given a *connected* graph G as input, lists **all** the triangles in G. (It is fine to list one triangle more than once.) We call this the `triangle listing problem` (duh!). You can assume that as input you are given G in *both* the adjacency matrix and adjacency list format. *For this problem you can also assume that G is connected.*

Sample Input/Output pair

- `Input:` Consider the input $G=(\{A,B,C,D\},\{(A,B),(A,C),(A,D),(B,C),(B,D),(C,D)\})$: i.e. a graph on 4 vertices where all possible $\binom{4}{2}=6$ pairs of vertices are connected by an edge.
- `Output:` The list $\{A,B,C\},\{A,B,D\},\{A,C,D\},\{B,C,D\}$.

Below are the two parts of the problem:

- Part `(a)` : Present an $O(n^3)$ time algorithm to solve the triangle listing problem.

  > **Note**
  > The trivial algorithm works. For this part only a correct algorithm idea is needed.

- Part `(b)` : This has two sub-parts:
  1. Let Δ denote the maximum degree, i.e.

     $$\Delta = \max_u n_u.$$

     Present an $O(n \cdot \min(m, \Delta^2))$ time algorithm to solve the triangle listing problem.

     > **Hint**
     > It might be useful to consider all the potential triangles that have one fixed vertex $u \in V$ as one of its vertices.

     > **Note**
     > Both algorithm idea and details are needed. Only proof idea (for correctness of the algorithm) is needed: no proof details are required. Runtime analysis is also needed.

  2. Present an $O(m^{3/2})$ algorithm to solve the triangle listing problem.

     > **Note**
     > Only a valid algorithm idea and runtime analysis is required.

Walkthrough video

Submission

> **!** PDF only please
>
> Autolab might not be able to display files in formats other than PDF (e.g. Word cannot be displayed). **If Autolab cannot display your file, then you will get a zero (0) on the entire question. Note that Autolab will NOT give an error message if you submit non-PDF file, so it is YOUR responsibility to make sure you submit in the correct format.** Also the file size has to be **at most 3MB**.

Grading Guidelines

We will follow the usual grading guidelines for non-programming questions (../../policies/hw-policy.html#grading). Here is a high level grading rubric specific to part `(a)` of this problem:

- `Algorithm idea` : `10` points.

and here is the high level grading rubric for part `(b)` :

- Part 1:
  1. `Algorithm idea` : `2` points.
  2. `Algorithm details` : `2` points.
  3. `Proof idea` : `2` points.
  4. `Runtime analysis` : `2` points.
- Part 2:
  1. `Algorithm idea` : `4` points.
  2. `Runtime analysis` : `3` points.
- `Note` : If you only attempt part `(b)` , then your score for part `(b)` will be scaled down to out of a max of `10` points for your score for part `(a)` .

> **!** Note
>
> **If you do not have labeled separated out parts as mentioned in the grading rubric above, you will get a zero(0) irrespective of the technical correctness of your solution**.

> Templates

Download LaTeX template. (latex-template.zip)

Download Microsoft Word template. (word-template.zip)

! Note

Your must explicitly list your sources and collaborators when you upload your submission to Autolab. Note that there are only five allowed sources (../../policies/hw-policy.html). If you have used a source that is not allowed, please do **not** submit your homework. If you did not consult any source or did not collaborate with anyone just say `None` .